La programmazione in linguaggio

BASIC

Il BASIC del sistema AMICO 2000

il
BASIC
del sistema

ADVANCED MICROCOMPUTER SYSTEM

SOMMARIO

1

INTRODUZIONE

Prima che un computer possa fare qualcosa di utile, bisogna "dirgli" cosa fare. Sfortunatamente, oggi come oggi, i computer non sono in grado di capire nè l'italiano, nè nessun altro linguaggio "umano". Questo perchè il nostro linguaggio è pieno di ambiguità e sottointesi. Al computer bisogna invece dare delle istruzioni precise e l'ordine esatto in cui eseguirle per ottenere il risultato desiderato. Perciò, per facilitare le comunicazioni uomo-macchina, sono stati sviluppati dei linguaggi di programmazione.

Il BASIC è un linguaggio di programmazione facile da capire e semplice da usare. È un ottimo strumento per applicazioni in campi quali l'economia, le scienze e l'insegnamento. Dopo aver usato il BASIC solo poche ore, vi accorgerete di essere già in grado di scrivere programmi con una facilità che pochi altri linguaggi di programmazione permettono.

Sviluppato inizialmente all'Università di Dartmouth, il BASIC ha trovato ampio consenso nel mondo dei computer. Benchè sia uno dei linguaggi piú facili da usare, è molto potente. Il BASIC usa un ristretto gruppo di normali parole inglesi come "comandi". Essendo progettato come linguaggio interattivo gli potete dare un comando come "PRINT 2 + 2" ed il BASIC vi risponderà immediatamente "4". Non è necessario fargli leggere il vostro programma perforato su un pacco di schede e poi aspettare il risultato per ore. Al contrario, avrete tutta la potenza del computer "sulla punta delle dita". Speriamo che apprezziate il BASIC e che riusciate a risolvere i vostri problemi di programmazione.

2

PRIMO APPROCCIO AL BASIC

Questo capitolo non intende essere una spiegazione dettagiata della programmazione in BASIC. Servirà piuttosto da introduzione a coloro i quali non hanno familiarità col linguaggio.

Consigliamo di provare ogni esempio di questo capitolo cosí come è presentanto. Questo migliorerà la vostra "sensibilità" per il BASIC e per il suo uso. Nella tabella 2-1 sono elencati tutti i comandi del BASIC. Una volta entrati nel BASIC, ogni volta che il computer visualizza OK si può battere uno statement in BASIC. Tutti i comandi impartiti al BASIC devono terminare con il carattere di ritorno al carrello (CR).

Il ritorno carrello dice al BASIC che avete finito di battere il comando. Se fate un errore di battitura, battete CTRL H per cancellare l'ultimo carattere. L'uso ripetuto di CTRL H elimina i caratteri precedenti.

L'at-sign (@) cancella tutta la riga, e CTRL L cancella tutto lo schermo.

TABELLA 2-1 - COMANDI DEL BASIC

COMANDI
(Par.3-3)

CLEAR
CONT
FRE
LIST
LOAD
NEW
PEEK
POKE
RUN
SAVE

STATEMENTS DI PROGRAMMA
(Par.3-4)

DEF FN
DIM
END
FOR
GOSUB
GOTO
IF...GOTO
IF...THEN
LET
NEXT
ON...GOSUB
ON...GOTO
REM
RESTORE
RETURN
STOP
USR
WAIT

STATEMENT DI IN/OUT
(Par. 3-5)

DATA
GET
INPUT
READ
PRINT
SPC
TAB

FUNZIONI SU STRINGHE
(Par.3-6)

ASC
CHR$
LEFT$
LEN
MID$
RIGHT$
STR$
VAL

FUNZIONI ARITMETICHE
(Par. 3-7)

ABS
ATN
COS
EXP
INT
LOG
RND
SIN
SGN
SQR
TAN
PGR

COMANDI DIRETTI

Cominciamo a battere:

PRINT 10-4 (terminate con il ritorno del carrello)

Il BASIC stamperà immediatamente:

6
OK

Il comando di stampa è stato eseguito appena avete premuto il tasto di ritorno del carrello. Questo si chiama un comando diretto. Il BASIC ha calcolato la formula dopo il "PRINT" e poi stamperà il suo valore, in questo caso "6".

Ora provate a battere:

PRINT 1/2,3 * 10 ("*" significa moltiplicato, "/" significa diviso)

Il BASIC stamperà:

.5 30

Come potete vedere, il BASIC può moltiplicare e dividere oltre che sottrarre. Notate come una "," (virgola) sia stata usata nel comando di stampa per far stampare due valori invece che uno solo.

Il comando divide la riga di 64 caratteri in 4 colonne, larghe 16 caratteri. La "," segnala al BASIC di posizionare il carrello del terminale all'inizio del successivo gruppo di 16 caratteri, dove viene stampato il valore 30.

COMANDI INDIRETTI

C'è un altro tipo di comando, chiamato comando indiretto. Ogni comando indiretto comincia con un numero di riga.

Un Numero di Riga è un qualsiasi intiero tra 0 e 63999.

Provate a battere queste righe:

```
10 PRINT 2+3
20 PRINT 2-3
```

Una sequenza di comandi indiretti si chiama un "programma". Invece di eseguire gli statements indiretti subito, il BASIC li memorizza. Quando battete RUN, il BASIC eseguirà dapprima lo statement indiretto col numero di riga piú basso, poi quello subito piú alto ecc., fino ad avere esaurito gli statements memorizzati.

Nell'esempio precedente, abbiamo battuto prima la riga 10 e poi la riga 20. In ogni modo, non importa in quale ordine vengono battuti gli statements. Il BASIC li mette sempre nell'ordine giusto secondo il numero di riga.

Immaginate di battere:

RUN

Il BASIC stamperà:

5
-1

Nel paragrafo 2-2 abbiamo memorizzato un programma di 2 righe. Vediamo ora come si può usare il BASIC per operare su una o entrambe le righe.

STAMPA DEL PROGRAMMA

Se vogliamo la stampa di tutto il programma attualmente in memoria, basta battere:

```
LIST
```

Il BASIC risponderà:

```
10 PRINT 2+3
20 PRINT 2-3
```

CANCELLAZIONE DI UNA RIGA

Qualche volta sorge la necessità di cancellare una riga di programma. Si ottiene questo battendo il Numero di Riga della riga da cancellare seguito dal ritorno del carrello.

Battete:

```
10
LIST
```

Il BASIC vi risponderà:

```
20 PRINT 2-3
```

Abbiamo cosí cancellato la riga 10 del programma.

SOSTITUZIONE DI UNA RIGA.

È possibile sostituire la riga 10, invece che cancellarla, battendo la nuova riga 10 seguita dal ritorno del carrello.

Battendo:

```
10 PRINT 3-3
LIST
```

Il BASIC risponderà:

```
10 PRINT 3-3
20 PRINT 2-3
```

Sconsigliamo di dare alle righe numeri consecutivi. Potrebbe diventare necessario inserire una riga nuova tra due già esistenti. Un incremento di 10 tra numeri di riga è generalmente sufficiente.

CANCELLARE UN PROGRAMMA

Se volete cancellare tutto il programma in memoria, battete "NEW". Se avete finito di lavorare con un programma e state per caricarne uno nuovo, ricordatevi di battere prima "NEW".

Battete:

NEW

Ora battete:

LIST

STAMPA DEI DATI.

Spesso è desiderabile includere un testo insieme alle risposte che vengono stampate, per spiegare il significato dei numeri.

PRINT "UN TERZO È PARI A", 1/3

Il BASIC vi risponderà:

UN TERZO È PARI A .333333333

Come spiegato al paragrafo 2-2 la "," nello statement PRINT fa passare il carrello al successivo gruppo di 10 colonne prima di stampare il valore che segue la ",". Usando un ";" al posto della virgola, il valore successivo verrà invece stampato subito dopo quello precedente.

NOTA: i numeri vengono sempre stampati preceduti da almeno uno spazio. Qualsiasi testo da stampare deve essere sempre messo tra virgolette.

Provate i seguenti esempi:

1. PRINT "UN TERZO È PARI A"; 1/3
   UN TERZO È PARI A .333333333
2. PRINT 1, 2, 3
   1 2 3
3. PRINT 1; 2; 3
   1 2 3
4. PRINT -1; 2; -3
   -1 2 -3

FORMATO DEI NUMERI

Questa breve digressione per spiegare il formato dei numeri del BASIC. I numeri vengono memorizzati interamente con piú di nove cifre di precisione. Quando un numero viene stampato, vengono stampate solo nove cifre. Ogni numero può anche avere un esponente (una potenza di dieci per quale è moltiplicato). Il numero piú grande che può essere rappresentato dal BASIC è 1.70141 * 10^{38}, mentre il piú piccolo numero positivo è 2.93874 * 10^{-39}.

1. Se il numero è negativo, viene stampato un segno (-).
   Se il numero è positivo viene stampato uno spazio.
2. Se il valore assoluto del numero è un intero compreso tra lo 0 e 999999999, viene stampato come intero.
3. Se il valore assoluto del numero è minore o uguale a 999999999, viene stampato in notazione a punto decimale fisso, senza esponente.
4. Se il numero non rientra nella categoria 2 o 3, viene usata la notazione scientifica.

La notazione scientifica è cosí formata:

SX.XXX XXX XXE STT (ogni X è un intero tra 0 e 9).
La prima "S" è il segno del numero: uno spazio per i numeri positivi, e un "-" per quelli negativi. Una cifra diversa da zero viene stampata prima del punto decimale.
Questa è seguita dal punto decimale e poi dalle altre otto cifre della mantissa. Segue una "E" (che sta per esponente), seguita dal segno dell'esponente (S) dalle due cifre (TT) dell'esponente stesso.
Gli zeri nelle posizioni piú significative non vengono mai stampati: la cifra a sinistra del punto decimale non è mai zero. Anche gli zeri superflui a destra non vengono stampati. Se resta una sola cifra da stampare dopo che sono stati eliminati tutti gli zeri non significativi, non viene stampato il punto decimale. Il segno dell'esponente è " + " per gli esponenti positivi, e "-" per quelli negativi.
Vengono sempre stampate due cifre per l'esponente; gli zeri non vengono soppressi nel campo dell'esponente. Il valore di ogni numero è perciò il numero alla sinistra della "E" moltiplicato per dieci elevato alla potenza indicata dal numero alla destra della "E".
Qualsiasi formato venga usato, viene sempre stampato uno spazio dopo il numero.
Il BASIC controlla per vedere se tutto il numero ci sta nella riga attuale. Se non ci sta, viene eseguito un "a capo" prima di stampare il numero.

Ecco gli esempi di vari numeri e il formato d'uscita in cui il BASIC li stamperà:

NUMERI	FORMATO D'USCITA
+1	1
-1	-1
6523	6523
-23.460	-23.46
1E20	1E+20
-12.3456E-7	-1.23456E.06
1.234567E-10	1.23457-10
1000000000	1E+09

NUMERI	FORMATO D'USCITA
999999999	999999999
.1	.1
.01	01
.000123	1.23E-04

Un numero introdotto dal terminale o una costante numerica usata in un programma BASIC può avere quante cifre si vuole, fino alla lunghezza di una riga (64 caratteri). Comunque solo le prime 10 cifre sono significative, e la decima viene arrotondata.

```
PRINT 1.23456789876543210
1.23456789
```

ASSEGNAMENTO DI VARIABILI CON UN STATEMENT DI INGRESSO

Ecco un esempio di un programma che legge un valore dal terminale e lo usa per calcolare e stampare un risultato:

```
10 INPUT R
20 PRINT 3.14159*R*R
RUN
? 10
314.159
```

Il funzionamento è il seguente: quando il BASIC incontra lo statement di ingresso (INPUT), stampa un punto interrogativo (?) e aspetta che voi battiate un numero. Dopodichè (nell'esempio è stato battuto 10) alla variabile (R) viene assegnato il valore (10) e l'esecuzione continua con lo statement sucessivo.

Quando viene calcolata la formula dopo lo statement PRINT il valore 10 viene sostituito alla variabile R ogni volta che R compare nella formula diventa 3.14159 * 10 * 10, cioè 314.159.

Se volessimo calcolare l'area di vari cerchi, potremo rieseguire il programma per ogni cerchio. Ma c'è un modo piú semplice: basta aggiungere un'altra riga al programma; cosí:

```
30 GOTQ 10
RUN
? 10
314.159
? 3
28.2743
? 4.7
69.3977
?
```

Mettendo uno statement di salto ("GOTO") alla fine del programma, l'abbiamo fatto ritornare alla riga 10 dopo la stampa della risposta per ogni cerchio. Si poteva andare avanti all'infinito, ma abbiamo deciso di fermarci dopo aver calcolato l'area di tre cerchi. Per fare questo è stato battuto un ritorno del carrello allo statement di ingresso (cioè una riga bianca).

I NOMI DELLE VARIABILI

La lettera "R" che abbiamo appena usato nel programma si chiama una "variabile". Il nome di una variabile può essere un qualsiasi carattere alfabetico e può essere seguito da un qualsiasi carattere alfanumerico (lettere dalla A alla Z, numeri da 0 a 9).

Qualsiasi carattere alfanumerico dopo i primi due viene ignorato.
Ecco gli esempi di nomi di variabili validi e non:

VALIDI	NON VALIDI
A	% (il 1 carattere deve essere alfabetico)
Z1	ZIABCD (nome di variabile troppo lungo)
TP	TO (i nomi di variabili non possono essere parole riservate)
PSTGS	
COUNT	RGOTO (i nomi di variabili non possono contenere parole riservate).

ASSEGNAZIONE DI VALORI CON STATEMENT "LET".

Oltre ad assegnare i valori alle variabili con uno statement di ingresso, è possibile farlo con un LET o statement di assegnazione.

Provate il seguente esempio:

```
A =5
PRINT A,A*2
5       10
LET   Z = 7
PRINT Z,Z - A
7       2
```

Come noterete dagli esempi, il "LET" è opzionale in uno statement di assegnazione.
Il BASIC "ricorda" i valori che sono stati assegnati alle variabili usando questo tipo di statement. Questo "ricordarsi" adopera spazio di memoria per mettervi i dati.

I valori delle variabili vengono persi e lo spazio loro riservato in memoria viene rilasciato quando si verifica una delle seguenti situazioni.

- viene battuta una nuova riga nel programma o ne viene cancellata una vecchia
- viene dato un comando di CLEAR
- viene dato un comando di RUN
- viene battuto NEW

Un altro fatto importante è che se una variabile viene incontrata in una formula prima che le venga assegnato un valore, le viene automaticamente assegnato il valore zero.

Questo zero viene poi sostituito come valore della variabile nella formula stessa.

Ad esempio:

```
PRINT Q; Q+2;Q*2
0    2    0
```

PAROLE RISERVATE

Le parole usate come statements del BASIC sono "riservate" per scopi specifici. Non è permesso usare queste parole come nomi di variabili o all'interno di nomi di variabili. Per esempio "FEND" non è valido perchè "END" è una parola riservata.

La tabella 2-6.1 è la lista delle parole riservate dal BASIC:

TABELLA 2-6.1 PAROLE RISERVATE DEL BASIC.

ABS	FN	LIST	PRINT	SPC
AND	FOR	LOAD	POS	SQR
ATN	FRE	LOG	READ	STEP
ASC	GET	MID$	REM	STOP
CHR$	GOSUB	NEW	RESTORE	STR$
CLEAR	GOSUB	NEXT	RETURN	TAB
CONT	IF	NOT	RIGHT$	TAN
COS	INPUT	NULL	RND	THEN
DATA	INT	ON	RUN	TO
DEF	LEFT$	OR	SAVE	USR
DIM	LEN	PEEK	SGN	VAL
END	LET	PGR	SIN	WAIT
EXP		POKE		

COMMENTI

Lo statement REM (abbreviazione di "remark", commento) viene usata per inserire commenti o note in un programma. Quando il BASIC Incontra lo statement REM, il resto della riga viene ignorato.

Questo serve soprattutto come aiuto per il programmatore e non ha alcuna influenza sul funzionamento del programma nel risolvere un particolare problema.

TEST DI CONFRONTO

Supponete di dover scrivere un programma che verifichi se un numero è zero. Con gli statements finora illustrati ciò non è possibile. Ci vuole uno statement che può essere usato per saltare condizionatamente ad un altro statement. Lo statement "IF-THEN" (= se-allora) fa proprio questo.

Battete il seguente programma (ricordatevi di battere NEW prima).

```
10 INPUT B
20 IF B = 0 THEN 50
30 PRINT "NON-ZERO"
40 GOTO 10
50 PRINT "ZERO"
60 GOTO 10
```

Quando questo programma viene introdotto nel computer e fatto girare, chiede un valore per B. Battete un valore qualsiasi. L'AMICO 2000 arriverà poi allo statement "IF THEN". Tra la parola IF e THEN dello statement ci sono due espressioni separate da un segno di "relazione". Un segno di relazione è uno dei seguenti sei simboli:

SEGNO DI RELAZIONE	SIGNIFICATO
=	uguale a
>	maggiore di
<	minore di
< >	diverso da
< = o = <	minore o uguale a
> = o = >	maggiore o uguale a

Lo statement IF è vero o falso a seconda che le due espressioni soddisfino o meno la relazione. Per esempio, nel programma appena fatto, se viene battuto 0 come valore per B lo statement IF è vero perchè 0 = 0. In questo caso, poichè il numero dopo THEN è 50, l'elaborazione del programma continua alla riga 50. Perciò viene stampato "ZERO" e il programma salta indietro alla riga 10 (a causa dello statement GOTO alla riga 60).

Supponete ora di battere 1 come valore per B. Poichè 1 = 0 è falso, lo statement IF è falso e il programma continua l'esecuzione alla riga successiva. Perciò viene stampato "NON ZERO" e il GOTO alla riga 40, rimanda il programma indietro alla riga 10.

Adesso provate il seguente programma che confronta due numeri:

```
10 INPUT A,B
20 IF A < = B THEN 50
30 PRINT "A È MAGGIORE"
40 GOTO 10
50 IF A < B THEN 80
60 PRINT "SONO UGUALI"
70 GOTO 10
80 PRINT "B È MAGGIORE"
90 GOTO 10
```

Quando viene fatto girare questo programma, la riga 10 accetterà due numeri dal terminale. Alla riga 20, se A è maggiore di B, A < = B è falso. Questo fa eseguire lo statement successivo che stampa "A È MAGGIORE", e poi la riga 40 rimanda il programma indietro alla riga 10 a ricominciare.

Alla riga 20, se A ha lo stesso valore di B, A < = B è vero, perciò si passa alla riga 50. Alla riga 50, poichè A ha lo stesso valore di B, A < B è falso; perciò si passa allo statement successivo e viene stampato "SONO UGUALI". Infine la riga 70 rimanda l'esecuzione all'inizio.

Alla riga 20, se A è piú piccolo di B, A < = B è vero, perciò si passa alla riga 50. Alla riga 50, A < B è pure vero, perciò si passa alla riga 80, dove viene stampato "B È MAGGIORE" e poi si ritorna all'inizio.

Provate gli ultimi due programmi un po' di volte. Vi sarà piú facile capire se provate a scrivere un vostro programma usando lo statement IF-THEN. In effetti provare programmi propri è la maniera piú facile e veloce per capire come funziona il BASIC.

Ricordatevi che per terminare l'esecuzione di questi programmi è sufficiente battere un ritorno carrello come risposta allo statement di ingresso.

ANELLI

Uno dei vantaggi dei computer è la loro abilità nell'eseguire compiti ripetitivi, vediamo come esempio un programma che esegue la radice quadrata.

Immaginate di volere una tabella delle radici quadrate dei numeri da 1 a 9. La funzione del BASIC che calcola la radice quadrata è "SQR". Viene usata nella forma "SQR (X)", dove X è il numero di cui si vuole la radice. Potremo scrivere il programma così:

```
10 PRINT 1; SQR (1)
20 PRINT 2; SQR (2)
30 PRINT 3; SQR (3)
40 PRINT 4; SQR (4)
50 PRINT 5; SQR (5)
60 PRINT 6; SQR (6)
70 PRINT 7; SQR (7)
80 PRINT 8; SQR (8)
90 PRINT 9; SQR (9)
```

Questo programma otterrebbe lo scopo, ma sarebbe molto poco efficiente. Potremmo migliorare considerevolmente il programma usando lo statement IF appena introdotto:

```
10 N=1
20 PRINT N; SQR (N)
30 N=N+1
40 IF N <= 9 THEN 20
```

Quando viene fatto girare questo programma, la sua uscita sarà esattamente uguale a quella del programma precedente. Il programma funziona così:
la riga 10 è uno stratement LET che inizializza il valore di N = a 1. La riga 20 stampa N e la radice quadrata di N, usandone il valore attuale. Perciò equivale a 20 PRINT 1; SQR (1), e il risultato di questi calcoli viene stampato. Alla riga 30 c'è uno statement LET apparentemente molto strano.

Matematicamente scrivere N = N + 1 è senza senso. D'altra parte è importante ricordare che nello statement LET il simbolo " = " non significa uguaglianza. In questo caso " = " significa "sostituisci con". Tutto quello che fa questo statement è aggiungere 1 al valore attuale di N. Perciò dopo la prima esecuzione della riga 30, N vale 2.

Alla riga 40, poichè adesso N vale 2, N<9 è vero, pertanto la parte THEN fa riprendere l'esecuzione alla riga 20, con N che vale 2.

Il risultato è che vengono ripetute le righe dalla 20 alla 40, ogni volta incrementando di 1 il valore di N. Finalmente quando N alla riga 20 vale 9, la riga 30 lo incrementa a 10.

Accade perciò che il confronto alla riga 40 dà risultato falso, e poichè non ci sono altri statements il programma si ferma.

GLI STATEMENTS DEL BASIC PER LA COSTRUZIONE DI ANELLI

Questa tecnica si chiama "iterazione". Poichè in programmazione viene usata abbastanza spesso, ci sono degli statements speciali nel BASIC per usarla. Li introduciamo con un esempio:

```
10 FOR N = 1 TO 9
20 PRINT N; SQR (N)
30 NEXT N
```

L'uscita di questo programma è esattamente uguale a quella dei due programmi precedenti.

Alla riga 10, N viene inizializzato a 1. La riga 20 fa stampare il valore di N e la sua radice quadrata.

Alla riga 30 troviamo uno statement nuovo. "NEXT N" (letteralmente, "il prossimo N") aggiunge uno a N, e poi se N < =9 si ritorna allo statement che segue lo statement "FOR". Nel complesso perciò il programma lavora come quello precedente.

Notate che la variabile che segue il "FOR" è esattamente la stessa che segue il "NEXT". Al posto della N si sarebbe potuta usare una variabile qualsiasi; basta che sia la stessa sia nello statement "FOR" che nel "NEXT".

Per esempio si potrebbe sostituire "ZI" ogni volta che compare "N" e il programma funzionerebbe alla stessa maniera.

Supponiamo ora di voler stampare una tabella delle radici quadrate dei numeri pari fra 10 e 20. Si potrebbe scrivere il programma così:

```
10 N = 10
20 PRINT N; SQR (N)
30 N = N+2
40 IF N < = 20 THEN 20
```

Notate come questo programma sia simile alla versione "migliorata" del programma per stampare le radici dei numeri da 1 a 9. Questo stesso programma può essere scritto usando anche lo statement "FOR" appena introdotto:

```
10 FOR N = 10 TO 20 STEP 2
20 PRINT N; SQR (N)
30 NEXT N
```

L'unica differenza tra questo programma e quello precedente impiegante lo statement "FOR" è l'aggiunta delle parole "STEP 2" (letteralmente "con passo 2").

Questo dice al BASIC di incrementare N di 2 ogni volta, invece che di 1. Se nello statement "FOR" non è specificato lo "STEP", il BASIC aggiunge 1 ogni volta.

Lo "STEP" può essere una espressione qualunque.

Supponete ora di voler contare all'indietro, da 10 a 1. Ecco un programma adatto allo scopo:

```
10 I = 10
20 PRINT I
30 I = I-1
40 IF 1 > = I THEN 20
```

Notate che ora il confronto viene fatto per vedere se 1 è maggiore o uguale al valore finale. La ragione è che stiamo contando all'indietro. Nell'esempio precedente invece era il contrario, controllavamo che la variabile fosse minore o uguale al valore finale.

Lo "STEP" può essere usato anche con numeri negativi per ottenere lo stesso scopo. Questo può essere ottenuto usando gli stessi statements di prima:

```
10 FOR I=10 TO 1 STEP -1
20 PRINT I
30 NEXT I
```

Gli anelli ottenuti con i "FOR" possono venire anche messi all'interno di altri anelli: per esempio:

```
10 FOR I = TO5
20 FOR J = TO3
30 PRINT I,J
40 NEXT J
50 NEXT I
```

Notate che "NEXT J" procede "NEXT I", perchè l'anello di J è dentro l'anello di I. Il programma che segue invece è sbagliato, provate a farlo girare e vedrete cosa succede;

```
10 FOR I=1 TO5
20 FOR J=1 TO3
30 PRINT I,J
40 NEXT I
50 NEXT J
```

Non funziona perchè quando viene incontrato "NEXT I", viene persa l'informazione relativa all'anello di J.

Questo perchè l'anello di J è interno a quello di I.

Par. 2-9

OPERAZIONI SU MATRICI

Spesso è necessario poter scegliere un elemento qualsiasi in una tabella di numeri. Il BASIC permette questo mediante l'uso di matrici.

Una matrice è una tabella di numeri. Il nome di questa tabella (nome della matrice) è un qualsiasi nome valido di variabile, per esempio "A". Il nome di matrice "A" è distinto e separato dal nome di variabile semplice "A", ed è permesso usare entrabi in uno stesso programma.

Per scegliere un elemento della tabella, diamo un indice ad "A": per scegliere l'I-esimo elemento, ad "A" facciamo seguire I chiuso tra parentesi (I).

Perciò, "A(I)" è l'I-esimo elemento della matrice "A".

"A(I)" è solo un elemento della matrice "A", e al BASIC bisogna dire quanto spazio allocare per tutta la matrice. A questo scopo esiste lo statement "DIM". Per esempio, "DIM A (15)" significa riservare abbastanza spazio perchè "I" possa variare tra 0 e 15. Gli indici delle matrici partono sempre da 0; perciò nell'esempio precedente, abbiamo predisposto spazio per 16 numeri per la matrice A.

Se "A (I)" viene usata in un programma prima di essere dimensionata, il BASIC le riserva spazio per 11 elementi (da 0 a 10).

Come esempio dell'uso di matrici, provate il seguente esempio per ordinare una lista di 8 numeri, nel quale voi date i numeri da ordinare:

```
 10 DIM A (8)
 20 FOR I = 1 TO 8
 30 INPUT A (I)
 50 NEXT I
 70 F = 0
 80 FOR I = 1 TO 7
 90 IF A(I) < = A(I + 1) THEN 140
100 T = A (I)
110 A(I) = A(I + 1)
120 A(I + 1) = T
130 F = 1
140 NEXT 1
150 IF I = 1 TO 8
160 FOR I = 1 THEN 70
170 PRINT A(I)
180 NEXT I
```

Quando viene eseguita la riga 10, il BASIC riserva spazio per 9 numeri, da A(0) a A(8).

Le righe dalla 20 alla 50 prendono la lista non ordinata dall'utente. L'ordinamento stesso viene ottenuto percorrendo la lista e scambiando le coppie che non sono in ordine. "F" viene usato per indicare se è stato

fatto almeno uno scambio. Se si, la riga 50 dice al BASIC di tornare indietro e ripercorrere la lista cercando altre coppie non in ordine.

Se nessun numero è stato scambiato, o dopo che sono stati messi tutti in ordine, le righe dalla 160 alla 180 stampano la lista ordinata.

Notate che un indice può essere una espressione qualsiasi.

SUBROUTINES

Se avete un programma che compie la stessa azione in molti punti differenti, potete duplicare gli stessi statements riguardanti questa azione in ognuno dei punti del programma.

Gli statements "GOSUB" e "RETURN" possono venire usati per evitare queste duplicazioni. Quando incontra un "GOSUB" il BASIC salta alla riga il cui numero segue "GOSUB". Però memorizza il punto del programma dal quale è saltato.

Quando incontra lo statement "RETURN", il BASIC ritorna indietro allo statement che segue l'ultimo "GOSUB" eseguito. Osservate il seguente programma:

```
 10 PRINT "QUAL'È IL PRIMO NUMERO?"
 30 GOSUB 100
 40 T = N
 50 PRINT "QUAL'È IL SECONDO NUMERO?"
 70 GOSUB 100
 80 PRINT "LA SOMMA DEI DUE NUMERI È", T + N
 90 STOP
100 INPUT N
110 IF N = INT (N) THEN 140
120 PRINT "MI DISPIACE, IL NUMERO DEVE ESSERE UN IN-
    TERO. RIPROVATE"
130 GOTO 100
140 RETURN
```

Questo programma chiede due numeri (che devono essere interi) e poi stampa la loro somma. La subroutine in questo programma è costituita dalle righe da 100 a 130.

La subroutine chiede un numero e, se non è un intero, ne chiede uno nuovo. Continua a chiederne finchè non viene battuto in valore intero.

Il programma principale stampa "Qual'è il primo numero?", e poi chiama la subroutine per assegnare il valore a N.

Quando la subroutine ritorna (alla riga 40) il valore in ingresso viene assegnato anche alla variabile T, in modo che, quando la subroutine viene chiamata una seconda volta non venga perduto il valore del primo numero.

Viene poi stampato "Qual'è il secondo numero?", e il secondo valore viene accettato dalla seconda chiamata della subroutine. Quando la subroutine ritorna la seconda volta, viene stampato "La somma dei due numeri è", seguito dalla somma. T contiene il valore del primo numero introdotto e N quello del secondo.

ARRESTO DELL'ELABORAZIONE

Lo statement successivo nel programma, è uno statement "STOP". Questo arresta l'esecuzione del programma alla riga 90. Se non ci fosse stato nel programma lo statement di "STOP", saremmo "finiti" nella subroutine 100. Ciò è male perchè ci verrebbe richiesto di introdurre un nuovo numero. Se lo introducessimo, poi, la subroutine cercherebbe di ritornare; e poichè non era stata chiamata da un "GOSUB" si verifica un errore RG. Ogni "GOSUB" eseguito in un programma deve avere un corrispondente "RETURN" eseguito piú avanti. Naturalmente vale anche l'inverso: un "RETURN" deve essere incontrato solo se fa parte di una subroutine che è stata chiamata da un "GOSUB".

Sia "STOP" sia "END" possono venire usati per separare un programma dalle sue subroutines.

La differenza è che "STOP" stampa un messaggio che dice a quale linea lo "STOP" è stato incontrato.

IMMISSIONE DEI DATI

Immaginate di avere dei numeri nel vostro programma che non cambiano ogni volta che il programma viene eseguito, ma che vorreste poter cambiare se necessario.

Il BASIC contiene degli statements speciali "READ" e "DATA", per questo scopo.

Osservate il seguente programma:

```
 10 PRINT "INDOVINA UN NUMERO"
 20 INPUT G
 30 READ D
 40 IF D = -999999 THEN 90
 50 IF D<> G THEN 30
 60 PRINT "HAI INDOVINATO"
 70 END
 90 PRINT "HAI SBAGLIATO, RIPROVA"
 95 RESTORE
100 GOTO 10
110 DATA 1,393, -39,28, 391, -8,0,3.14,90
120 DATA 89,5,19,15,-34, -999999
```

Quando viene incontrato lo statement "READ", l'azione è simile a quella di uno statement di INPUT: solo che invece di ricevere un numero dal terminale, lo legge dagli statement "DATA".

La prima volta che viene richiesto un numero per un READ, viene letto il primo numero nel primo statement DATA. La seconda volta viene letto il secondo numero del primo statement. Quando è stato letto tutto il contenuto del primo statement DATA, viene usato il secondo statement DATA. I DATA vengono letti sempre sequenzialmente in questo modo, e ci può essere un numero qualsiasi di statement DATA nel vostro programma.

Questo programma è un piccolo gioco che consiste nel cercare di indovinare uno dei numeri contenuti negli statement DATA. Per ogni tentativo che viene fatto, il programma legge tutti i numeri contenuti nello statement DATA finchè ne trova uno uguale al tentativo che è stato battuto.

Se si cerca di leggere più numeri di quelli contenuti nello statement DATA, viene segnalato un errore di "fuori dai dati" (OD) (in inglese, OUT OF DATA). È per questo motivo che alla riga 40 il programma controlla se il dato letto è -999999. Questo infatti non è uno dei numeri da indovinare, ma viene usato come segnalatore per indicare che tutti i dati (tutte

le possibili risposte giuste) sono stati letti. Perciò se viene letto -999999, sappiamo che il tentativo era sbagliato.

Prima di ritornare alla riga 10 per un altro tentativo, bisogna fare in modo che il READ successivo ricominci dal primo dato.

Questo viene fatto dal "RESTORE". Dopo che è stato incontrato un "RESTORE", la lettura dei dati ricomincia dal primo numero nel primo statement DATA. Gli statement DATA possono essere messi in qualsiasi punto del programma.

Gli statement DATA vengono usati solo dagli statement READ. Ogni volta che vengono incontrati durante l'esecuzione di un programma, vengono ignorati.

STRINGHE DI CARATTERI

Una serie di caratteri si chiama "stringa". MICROCOMPUTER e QUESTO È UN TEST sono tutte stringhe. Come le variabili numeriche anche le variabili di tipo stringa possono assumere valori. I nomi delle variabili di tipo stringa si distinguono dalle variabili numeriche per un "$" dopo il nome della variabile.

Provate questo esempio:

```
A$ = "MICROCOMPUTER"
PRINT A$
RUN
MICROCOMPUTER
```

In questo esempio alla variabile-stringa A$ assegnamo il valore-stringa "MICROCOMPUTER". Notate che abbiamo messo la stringa di caratteri da assegnare ad A$ tra virgolette.

Adesso che abbiamo assegnato un valore alla variabile A$ possiamo trovare la lunghezza di questa variabile (cioè il numero di caratteri che contiene). Si fa cosí:

```
PRINT LEN (A$), LEN ("MICROCOMPUTER")
13              13
```

La funzione "LEN"dà come risultato il numero di cui è composta la stringa di caratteri. Il numero di caratteri per stringa può variare tra 0 e 255. Una stringa contenente 0 caratteri si chiama stringa "vuota". Prima che a una variabile stringa venga assegnato un valore nel corso del programma, questa viene inizializzata come stringa vuota.

Stampare una stringa vuota significa non stampare alcun carattere. Non viene nemmeno fatto avanzare il carrello (o il cursore) alla colonna successiva.

Provate questo esempio:

```
PRINT LEN (Q$);Q$;3
0         3
```

Un altro modo per creare una stringa è il seguente: Q$ = "". Si può assegnare a una variabile stringa la stringa vuota, in questo modo si libera dello spazio che può essere usato da una variabile stringa non vuota.

Spesso si vuole usare e manipolare solo una parte di una stringa. Adesso che abbiamo assegnato ad A$ il valore "MICROCOMPUTER", potremmo voler stampare solo i primi 5 caratteri di A$. Per fare questo si usa la funzione LEFT$ ad esempio:

```
PRINT LEFT$ (A$,5)
MICRO
```

"LEFT$" è una funzione che da una stringa ne compone un'altra formata solo dagli N caratteri piú a sinistra.

Ecco un altro esempio:

```
FOR I = 1 TO LEN (A$): PRINT LEFT$ (A$,I):NEXT I
M
MI
MIC
MICR
MICRO
MICROC
MICROCO
MICROCOM
MICROCOMP
MICROCOMPU
MICROCOMPUT
MICROCOMPUTE
MICROCOMPUTER
```

Poichè A$ è di 13 caratteri, questo anello viene eseguito con I = 1,2,3,.....12,13.

Al primo passaggio viene stampato solo il primo carattere, al secondo i primi due, e cosí di seguito fino alla stampa di tutti i caratteri.

Un'altra funzione operante su stringhe è "RIGHT$", che preleva gli N caratteri piú a destra. Provate a sostituire "RIGHT$" a LEFT$" nell'esempio precedente, e guardate cosa succede.

C'è anche una funzione che ci permette di prendere caratteri dal centro di una stringa. Provate questo esempio:

```
FOR N = 1 TO LEN (A$): PRINT MID$ (A$,N): NEXT N
MICROCOMPUTER
ICROCOMPUTER
CROCOMPUTER
ROCOMPUTER
OCOMPUTER
COMPUTER
OMPUTER
MPUTER
PUTER
UTER
TER
ER
R
```

"MID$" produce una stringa che comincia con l'N-esimo e termina con l'ultimo carattere di A$. Il primo carattere della stringa e quello in posizione 1 e l'ultimo carattere possibile è quello in posizione 255.

Molto spesso si vuole estrarre solo l'N-esimo carattere di una stringa. Per fare questo, bisogna chiamare MID$ con tre argomenti. Il terzo argomento specifica il numero di caratteri desiderati.

Per esempio:

```
FOR N =  1 TO.LEN (A$): PRINT MID$ (A$,N,1), MID$ (A$,N,2): NEXT N
             M         MI
             I         IC
             C         CR
             R         RO
             O         OC
             C         CO
             O         OM
             M         MP
             P         PU
             U         UT
             T         TE
             E         ER
             R         R
```

Le stringhe possono venire anche concatenate (accostate) mediante l'uso dell'operatore " + ". Provate questo esempio:

```
B$ = "BASIC FOR" + " " + A$
PRINT B$
BASIC FOR MICROCOMPUTER
```

I concatenamenti sono molto comodi se dovete fare delle piccole modifiche a stringhe che avete già in memoria.

Per esempio:

```
CS = LEFT$ (BS,9) + "     " + MID$ (B$, 11, 5) + "     " + RIGHT$
               PRINT C
               BASIC FOR MICRO COMPUTER
```

Talvolta si vuole convertire un numero in stringa, o viceversa, "VAL" e "STR$" operano queste conversioni.
Provate questo esempio:

```
STRIN$ = "567.8"
PRINT VAL (STRING$)
567.8
STRING$ = STR$ (3.1415)
PRINT STRING$, LEFT$ (STRING$,5)
3.1415          3.14
```

"STR$" può essere usato per formattare i numeri in ingresso e in uscita. Convertendo un numero in stringa, e usando LEFT$, RIGHT$, MID$, e i concatenamenti, potete modificarne il formato a piacimento.

"STR$" può essere impiegato anche per sapere su quante colonne verrà stampato un numero. Per esempio:

```
PRINT LEN (STR$ (3.157))
6
```

Se invece l'utente pone al computer una domanda del tipo "QUAL'È IL VOLUME DI UN CILINDRO DI RAGGIO 5.36 METRI E ALTEZZA 5.1 METRI"? potete usare "VAL" per estrarre i valori numerici 5.36 e 5.1 dalla domanda.

CHR$ è una funzione operante su stringhe che ritorna con una stringa di un carattere che contiene l'equivalente ASCII dell'argomento, secondo la tavola di conversione di cui all'appendice E prende il primo carattere di una stringa e lo converte nel suo equivalente ASCII decimale. Uno degli usi piú comuni di CHR$ è quello di mandare un carattere speciale al terminale. Il piú usato di questi caratteri è BELL (ASCII 7). La "stampa" di questo carattere fa suonare un campanello sulla telescrivente o produce un "beep" sulla maggior parte dei terminali CTR.

Può essere usato come primo carattere di un messaggio di errore o comunque per richiamare l'attenzione dell'utente.
Esempio: PRINT CHR$ (7)

Una stringa può contenere da 1 a 255 caratteri. Tutte le variabili stringa terminano con il carattere dollaro ($), per ese., A$, B9$, K$, HELLO$.

Le matrici di stringhe possono venire dimensionate esattamente come le matrici numeriche. Per esempio DIM A (10, 10) produce una matrice di 121 stringhe, undici righe per undici colonne (righe da 0 a 10). Ogni elemento della matrice di stringhe è una stringa completa, che può essere lunga fino a 255 caratteri.

NOME	SCOPO/USO
DIM	Es. 27 LETA$ = "FOO" + V Assegna il valore di una espressione di tipo stringa a una variabile stringa. LET è opzionale. Operatori di confronto su stringhe. Il confronto viene fatto sulla base del codice ASCII, un carattere alla volta, fino a trovare una differenza. Se durante il confronto tra due stringhe, viene raggiunta la fine di una, la stringa piú corta viene considerata minore. Notate che "A" è piú grande di "A" poichè anche gli spazi terminali e iniziali sono significativi.

+	Es. 30 LETZ$ = R$ + Q$ Concatenamento di stringhe. La stringa risultante deve essere lunga meno di 256 caratteri, altrimenti verrà segnalato un errore di tipo LS.
INPUT	Es. 40 INPUT X$ Legge una stringa dal terminale dell'utente. Le stringhe non devono essere necessariamente tra virgolette; ma se non lo sono, vengono ignorati gli spazi iniziali e la stringa viene fatta terminare se viene incontrata una "," o un ":".
READ	Es. 50 READ X$ Legge una stringa da uno statement DATA contenuto nel programma. Le stringhe non devono essere necessariamente tra virgolette; ma se non lo sono, vengono interrotte alla prima virgola o al primo ":" e gli spazi iniziali vengono ignorati: vedi DATA per il formato dei dati di tipo stringa.
PRINT	Es. 60 PRINT X$ 70 PRINT "FOO" + A$ Stampa la stringa sul terminale dell'utente.

3

SPIEGAZIONE DEGLI STATEMENTS

Par. 3-1 CARATTERI SPECIALI

CARATTERE	USO
"	Cancella l'ultima carattere battuto. Se non ci sono piú caratteri sulla riga stampa un "ritorno carrello" e fa avanzare la carta di una riga.
CTRL I	Ė il duale di CTRL M: riscrive l'ultimo carattere cancellato. ATTENZIONE: non viene effettuato alcuno controllo per verificare se il cursore è oltre l'ultimo carattere valido. Se quindi cercate di riscrivere posizioni non cancellate otterrete caratteri a caso.
RITORNO CARRELLO	Un ritorno carrello deve terminare tutte le righe battute. Fa ritornare il cursore sulla posizione piú a sinistra della riga successiva, e stampa la riga se la stampante è abilitata.
CTRL S	Interrompe l'esecuzione di un programma o il comando LIST. CTRL S ha effetto al termine dell'esecuzione di uno statement o, nel caso del LIST, al termine della stampa di una riga. In entrambi i casi si ritorna all'accettazione dei comandi del BASIC, e viene stampato OK. Stampa "BREAK IN LINE XXXX" dove XXXX è il numero di riga dello statement che stava per essere eseguito.
: (due punti)	I due punti vengono usati per separare gli statements su una stessa riga. Possono venire usati sia negli statements diretti che in quelli indiretti. Il solo limite al numero si statements per riga è la lunghezza della riga.
?	Il punto di domanda equivale a PRINT. Per es., ?2 + 2 equivale a PRINT 2 + 2. Il punto di domanda può essere usato anche negli statement indiretti: 10? X nella stampa del testo di un programma viene stampato 10 PRINT X.

$	Mettere il suffisso $ al nome di una variabile significa dichiarare che la variabile è di tipo stringa.
%	Mettere il suffisso "per cento" (%) al nome di una variabile significa dichiarare che la variabile è intera.
!	Mettere il punto esclamativo (!) dopo INPUT, PRINT o ? fa stampare il dato in ingresso o in uscita anche se la stampante è disabilitata

ALT MODE	Ritorna il controllo al monitor.
CTRL P	Se la stampante dell'AMICO 2000 è abilitata la disabilita o viceversa.
CTRL L	Si comporta come @, ed in piú, cancella lo schermo.

OPERATORI DEL BASIC

SIMBOLO	SCOPO/USO
=	Es. A = 100 Assegna un valore a una variabile. ES. LET Z = 2.5 Il LET è opzionale.
—	Es. B = — A Negazione. Notate che 0—A è sottrazione, mentre —A è negazione.
∧ (SHIFT N)	130 PRINT X ∧ 3 Elevazione a potenza (nell'esempio è come X * X * X) 0∧0 = 1,0∧ a qualsiasi altra potenza = 0,A∧B, con A negativo e B non intero dà un errore di tipo FC
*	Es. 140 X = R (B * D) Moltiplicazione
/	Es. 150 PRINT X/ (1.3) Divisione.
+	Es. 160 Z = R + T + Q Somma.
—	Es. 170 J = 100—I Sottrazione.

REGOLE SEGUITE NEL CALCOLO DELLE ESPRESSIONI.

1) Gli operatori di priorità piú alta vengono applicati prima degli operatori di priorità piú bassa. Ciò significa che le moltiplicazioni e le divisioni vengono calcolate prima delle somme e sottrazioni. Per esempio, 2 + 10/5 fa 4, e non 2.4. Quando in una formula vengono incontrati operatori della stessa priorità quello piú a sinistra viene calcolato per primo: 6-3+5=8, e non -2.

2) L'ordine nel quale le operazioni vengono eseguite può essere sempre specificato mediante l'uso delle parentesi. Per esempio, per sommare 5 a 3 e poi dividere il risultato per 4, bisogna far calcolare (5+3)/4, che fa 2. Invece facendo calcolare 5+3/4 otterremmo 5.75 come risultato (5 piú 3/4).

L'ordine di precedenza degli operatori usati per calcolare le espressioni, cominciando da quello di precedenza piú alta è il seguente: (NOTA: gli operatori elencati sulla stessa riga hanno la stessa precedenza).

1)	Le espressioni tra parentesi vengono sempre calcolate prima.
2)	Elevamento a potenza.
3) NEGAZIONE	- X dove X può essere una formula.
4) * e /	Moltiplicazione e divisione.
5) + e —	Somma e sottrazione.

6) OPERATORI DI RELAZIONE
(stessa precedenza per tutti)

= uguale
< > diverso
< minore
> maggiore
< = o = < minore o uguale
> = o = > maggiore o uguale

(Questi che seguono sono operatori logici).

7) NOT	Negazione logica bit a bit. Come la negazione aritmetica, il not prende solo la formula alla sua destra come argomento.
8) AND	"AND" logico bit a bit.
9) OR	"OR" logico bit a bit.

Una espressione di relazione può essere usata come parte di qualsiasi espressione.

Le espressioni di operatori di relazione hanno sempre valore vero (1) oppure falso (0). Perciò, (5 = 4) = 0, (5 = 5) = 1), (4 > 5) = 0, (4 < 5) = 1, ecc.

La parte THEN di uno statement IF viene eseguita ogni qualvolta la formula dopo l'IF è diversa da zero. Pertanto, IF X THEN.... equivale a IF X = 0 THEN.......

SIMBOLO	SCOPO/USO
=	Es. 10 IF A = 15 THEN 40 Espressione uguale a espressione.
< >	Es. 70 IF A < > 0 THEN 5 Espressione diversa da espressione.
>	Es. 30 IF B > 100 THEN 8 Espressione maggiore di espressione.
<	Es. 160 IF B < 2 THEN 10 Espressione minore di espressione.
<=	180 IF 100 < = B + C THEN 10 Espressione minore o uguale a espressione.
=>	Es. 190 IF Q = > R THEN 50 Espressione maggiore o uguale a espressione.
AND	Es. 2F A<5 AND B<2 THEN 7 Se l'espressione I (A < 5) e l'espressione 2 (B < 2) sono entrambe vere, allora salta alla riga 7.
OR	Es. IF A < 1 OR B < 2 THEN 2 Se l'espressione I (A < 1) 0 L'espressione 2 (B < 2) è vera, salta alla riga 7.
NOT	Es. IF NOT Q3 THEN 4 Se l'espressione "NOT Q3" è vera (perchè l'espressione Q3 è falsa) allora salta alla riga 4. NOTA: NOT-1 = 0 (NOT vero = falso).

AND, OR e NOT possono essere usate per manipolare bit, e per eseguire operazioni BOOLEANE.

Questi tre operatori convertono i loro argomenti in numeri interi di sedici bit, in complemento a due, compresi tra -32768 e 32767. Eseguono poi le operazioni logiche specificate e ritornano con un risultato compreso nel medesimo campo. Se gli argomenti non sono compresi in detto campo, viene segnalato un errore di tipo "FC".

Le operazioni sono eseguite bit a bit: ogni bit del risultato viene ottenuto esaminando il bit nella stessa posizione di ciascun argomento.
La seguente tabella della verità mostra le relazioni logiche tra i bit:

OPERATORE	ARG. 1	ARG. 2	RISULTATO
AND	1	1	1
	0	1	0
	1	0	0
	0	0	0
OR	1	1	1
	1	0	1
	0	1	1
	0	0	0
NOT	1	—	0
	0	—	1

ESEMPI: (in tutti gli esempi qui sotto, non vengono scritti gli zeri non significativi iniziali).

63 AND 16 = 16 Poichè 63 in binario è 111111 e 16 in binario è 10000, il risultato dell'AND è il numero binario 10000, cioè 16.

15 AND 14 = 14 15 in binario è 1111 e 14 in binario è 1110, perciò i5 AND 14 è 1110 binario, cioè 14.

-1 AND 8 = 8 -1 in binario è 1111111111111111 e 8 è 1000, cosí il risultato è 1000 binario, cioè 8 decimale.

4 AND 2 = 0 4 in binario è 100 e 2 è 10, perciò il risultato in binario è 0, perchè nessuno dei bit nei due argomenti è messo in modo da dare un bit 1 nel risultato.

4 OR 2 = 6 L'OR di 100 binario e 10 binario da 110 binario che è esadecimale.

10 OR 10 = 10 L'OR di 1010 binario con 1010 binario dà ancora 1010 binario, cioè 10.

-1 OR -2 = -1 L'OR di 1111111111111111 binario è (-1) e 1111111111111111 (-2) dà 1111111111111111 binario, cioè -1.

NOT 0 = -1 Il complemento di ogni bit di 0 binario (a 16 bit) è 16 uni (1111111111111111) o -1. Inoltre NOT -1 = 0.

NOT X	NOT X è uguale a — (X + 1). Infatti per formare il completamento a due di un numero di sedici bit, si prende il complemento (a uno) dei bit, e si aggiunge uno.
NOT 1 = -2	Il complemento dei sedici bit di 1 è 1111111111111110, cioè è uguale a — (1 + 1) cioè -2.

Un tipico uso degli operatori che lavorano sui singoli bit è quello di controllare i bit nelle locazioni che riflettono lo stato di qualche dispositivo esterno. In una parola di memoria (byte) il bit X in posizione 7 è il piú significativo. Per esempio, supponiamo che il bit 1 della locazione 40963 è 0 quando la porta della stanza X è chiusa, 1 quando è aperta. Il seguente programma stampa "Attenzione: intruso" se la porta è aperta:

```
10 IF NOT (PEEK 40963 and 2) THEN 10
```

Questa riga viene continuamente rieseguita finchè il bit 1 (mascherato, selezionato dal 2) diventa 1. Quando ciò accade, si passa alla riga 20.

```
20 PRINT "ATTENZIONE: INTRUSO"
```

La riga 20 stampa "ATTENZIONE: INTRUSO". Naturalmente, possiamo sostituire lo statement 10 con uno statement "WAIT", che ha esattamente lo stesso effetto.

```
10 WAIT 40963, 2
```

Questa riga ritarda l'esecuzione dello statement successivo finchè il bit 1 della locazione A003 non diventa 1. Il WAIT è molto piú veloce dell'equivalente "IF" e richiede anche meno bytes per la memorizzazione del programma.

Ecco un altro utile impiego degli operatori di relazione:

```
125 A = - (B > C) * B - (B < =C) * C
```

Questo statement assegna alla variabile A il MAX (B,C) = la più grande tra le due variabili B e C.

COMANDI DEL BASIC

Un comando può venire dato solo dopo che il BASIC ha visualizzato il cursore. Questo si chiama il "Livello dei comandi". I comandi possono venire usati come gli statements dei programmi. Certi comandi, come LIST, NEW e LOAD terminano l'esecuzione dei programmi quando finiscono.

Ogni comando può richiedere uno o piú argomenti, oltre allo statements di comando, come spiegato nella descrizione della sintassi e delle funzioni.

Un argomento senza parentesi va battuto senza parentesi. Gli argomenti messi tra parentesi vanno messi tra le parentesi indicate. Gli argomenti messi tra parentesi quadre sono opzionali, dovete batterli a seconda dei casi con o senza le parentesi, come indicato.

STAT. SINTASSI/FUNZIONE

CLEAR Es. CLEAR
Azzera tutte le variabili di programma, resetta lo stato dei "FOR" e "GOSUB", reinizializza i dati.

CONT es: CONT
Continua l'esecuzione del programma dopo che è stato battuto CTRL S o è stato eseguito uno statement STOP o INPUT. Non è possibile continuare dopo un errore, dopo aver modificato il programma, o prima che il programma sia stato eseguito. Uno degli scopi principali di CONT è la ricerca degli errori. Supponete che dopo aver fatto girare il programma per un po' di tempo, non venga stampato nulla. Potrebbe darsi che i calcoli del programma sia caduto in un "anello infinito". Un anello infinito è una serie di statement in BASIC dai quali non c'è uscita. Il BASIC continua a eseguire la serie degli statement finchè voi non intervenite o viene spenta l'alimentazione.

Se sospettate che il programma sia in un anello infinito, tenete premuto CTRL S finchè viene visualizzato il messaggio BREAK. Viene stampato il numero dello statement che il BASIC stava eseguendo e visualizzato il cursore; potete usare PRINT per stampare i valori di alcune variabili. Dopo aver esaminato questi valori potrete essere sicuri che il programma funziona correttamente.

Allora potete battere CONT per continuare l'esecuzione del programma da dove era stato interrotto, o battere un GOTO per riprendere l'esecuzione direttamente a una riga diversa. Potete usare anche statement di assegnazione (LET) per assegnare ad alcune variabili valori differenti. Ricordatevi che se fermate un programma con CTRL S e intendete conti-

nuarlo piú tardi, non dovete incorrere in errori o aggiungere linee di programma.

Se lo fate, non potrete piú continuare, e vi sarà segnalato un errore di tipo "CN" (non continua). Non è possibile continuare un comando diretto.

CONT fa sempre riprendere l'esecuzione allo statement che stava per essere eseguito quando era stato battuto CNTRL S.

FRE — Es. 270 PRINT FRE

Dà il numero di bytes di memoria attualmente non usati dal BASIC. Se le locazioni libere sono superiori a 32767 vengono indicate con il complemento a 32768, ad esempio:
-32765 = 32767 + (32768-32765) locazioni libere, e cioè 32770.

LIST — LIST (linea di partenza)- (linea di arrivo).

Stampa il programma in memoria, eventualmente cominciando a una riga specificata. La stampa può venire interrotta mediante il tasto CTRL S (il BASIC finisce di stampare la riga). Ad esempio:

LIST	stampa tutto il programma
LIST 100	stampa solo la riga 100
LIST 100-1000	stampa le righe da 100 a 1000
LIST -1000	stampa dalla riga sotto esame fino alla riga 1000
LIST 100	stampa dalla riga 100 alla fine del programma.

NEW — Es. NEW

Cancella il programma in memoria e tutte le variabili.

PEEK — Es. 356 PRINT PEEK (1)

PEEK (indirizzo)
La funzione PEEK serve per conoscere il contenuto della cella di memoria 1 espresso in decimale. Tale contenuto può avere un valore compreso tra 0 e 255 inclusi. Se l'indirizzo è maggiore di 65535 o negativo, viene segnalato un errore di tipo FC. Un tentativo di leggere una locazione di memoria non esistente dà invece come risultato un numero imprevedibile.

POKE — Es. 357 POKE I,J

POKE locazione, byte
Lo statement POKE memorizza il byte specificato dal secondo argomento (J) nella locazione specificata dal primo argo-

mento (I). Il byte da memorizzare deve essere =0 e =255, altrimenti viene segnalato un errore FC.

ATTENZIONE: un uso incauto dello statement POKE potrebbe far funzionare male il vostro programma, il BASIC o le funzioni del monitor.

Notate che le pagine 0 e 1 della memoria sono riservate al BASIC e non vanno usate per memorizzare le variabili del programma dell'utente. Un POKE su una cella di memoria che non esiste non fa alcun danno.

Uno degli usi principali di POKE è quello di passare gli argomenti alle subroutines scritte in linguaggio macchina (V; appendice F). Potete usare PEEK e POKE, per esempio, anche per scrivere un programma che faccia la diagnosi della memoria, o un assembler.

RUN

Es. RUN 200
RUN (numero della riga)
Fa partire l'esecuzione del programma attualmente in memoria dallo statement specificato. RUN azzera tutte le variabili (esegue un CLEAR) e reinizializza i DATA. Se avete fermato il programma e volete continuare l'esecuzione a partire da qualche punto, usate uno statement diretto di GOTO per far riprendere l'esecuzione alla riga desiderata, o CONT per proseguire dopo un BREAK.

Fa partire l'esecuzione del programma RUN dallo statement recante il numero di riga piú basso.

STATEMENT DI PROGRAMMA

Nella descrizione degli statement che segue, un argomento B, C, V o W indica una variabile numerica, X denota una espressione numerica, X$ una espressione di tipo stringa, e I o J indicano una espressione che viene troncata all'intero precedente prima dell'esecuzione dello statement. Troncamento significa che la parte dopo il punto viene persa, cioè 3.9 diventa 3 e 4.01 diventa 4.

Una espressione è una serie di variabili, operatori, chiamate di funzioni e costanti che, dopo l'esecuzione delle operazioni e delle funzioni, esecuzione fatta osservando le regole di precedenza, dà come risultato un valore numerico o in una stringa.

Una costante può essere sia un numero (3.14) sia una stringa di caratteri ("F00").

STAT. SINTASSI/FUNZIONE

DEF Es. 100 DEF FNA (V) = V/B + C

DEF FN X (lista degli argomenti) = espressione. L'utente può definire funzioni come le funzioni già incorporate (SQR, SGN, ABS, ecc.) mediante l'uso dello statement DEF. Il nome della funzione è "FN" seguito da qualsiasi nome valido di variabile, per esempio: FNX, FNJ7, FNK0, FNR2. Le funzioni definite da utente non possono occupare piú di una riga. Una funzione può essere una espressione qualsiasi, ma può avere un solo argomento.

Nell'esempio B e C sono variabili che vengono usate nel programma. L'esecuzione dello statement DEF definisce la funzione. Le funzioni definite dell'utente possono essere ridefinite eseguendo un altro statement DEF per la stessa funzione. "V" si chiama variabile formale 'in inglese "dummy").

L'esecuzione di questo statement dopo quello riportato sopra farebbe assumere a Z il valore 3/B + C mentre il valore di V non viene toccato.

Es. 110 Z = FNA (3)

DIM Es. 113 DIM A(3), B(10)

DIM variabile (1 dimensione, 2 dimensione).

Predispone lo spazio per le matrici. Tutti gli elementi della matrice vengono azzerati dallo statement DIM.

Le matrici possono avere da una a 255 dimensioni.

Es. 114 DIM R3 (5,5) D$ (2,2,2)

Le matrici possono venire allocate dinamicamente durante l'esecuzione del programma. Se una matrice non viene dimensionata esplicitamente con uno statement DIM, si intende che sia monodimensionale, costituita da 11 elementi (da 0 a 10).

Es. 117 A(8) = 4

Se questo statement venisse incontrato prima di aver trovato uno statement DIM riferito ad A, sarebbe come se, prima dell'esecuzione della riga 117, fosse stato eseguito DIM A (10). Tutti gli altri indici partono da 0, perciò DIM X (100) in effetti alloca 101 elementi per la matrice.

END

Es. 999 END
Termina l'esecuzione del programma senza stampare un messaggio di BREAK (vedi STOP). CONT dopo uno statement END fa riprendere l'esecuzione dallo statement successivo all'END. END può essere usato in qualsiasi punto del programma, ed è opzionale.

FOR

Es. 300 FOR V = 1 TO 9.3 STEP 6
FOR variabile = espressione TO espressione (STEP espressione) (V. lo statement NEXT) V viene messo al valore dell'espressione che segue il segno di uguale in questo caso 1. Questo valore si chiama il valore iniziale.

Poi vengono eseguiti gli statements tra FOR e NEXT. Il valore finale è il valore dell'espressione che segue il TO. L'incremento è il valore dell'espressione che segue STEP. Quando viene incontrato lo statement NEXT, l'incremento viene aggiunto alla variabile.

Es. 310 FOR V = 1 TO 9.3

Se non viene specificato alcun incremento, questo viene assunto uguale a 1. Se l'incremento è positivo e il nuovo valore della variabile è = al valore finale (9.3 nell'esempio), o il valore dell'incremento è negativo e il nuovo valore della variabile è = al valore finale, viene eseguito il primo statement seguente il FOR.

Altrimenti viene eseguito lo statement successivo al NEXT. Tutti gli anelli FOR eseguono gli statement tra il FOR e il NEXT almeno una volta, anche nei casi come FOR V = I TO.

Notate che per il valore iniziale e finale e per l'incremento di un anello FOR si possono usare espressioni.
315 FOR V = 10* N TO 3.4/Q STEP SQR (R)

I valori delle espressioni vengono calcolati una volta sola, prima che venga eseguito il corpo dell'anello FOR... NEXT.

Es. 320 FOR V = 9 TO I STEP - 1

Quando viene eseguito lo statement successivo al NEXT, la variabile dell'anello non è mai uguale al valore finale, bensì al valore che ha fatto terminare l'esecuzione dell'anello FOR-NEXT. Gli statement tra il FOR e il corrispondente NEXT in entrambi gli esempi sopra riportati (310 e 320) vengono eseguiti nove volte.

Non usate la stessa variabile come indice per due anelli FOR-NEXT uno interno all'altro. Il numero di anelli FOR interni uno all'altro è limitato solo dalla memoria disponibile (V. appendice D).

GOSUB Es. 10 GOSUB 910
GOSUB numero della riga

Salta allo statement specificato (910) fino a che viene incontrato un "RETURN" dal quale viene fatto un salto allo statement successivo a GOSUB. Il numero di subroutines interne una all'altra è limitato solo dalla memoria disponibile.

GOTO Es. 50 GOTO 100
GOTO numero della riga.

Salta alla riga specificata.

IF...GOTO Es. 32 IF X < = Y + 23.4 GOTO 92
IF espressione GOTO numero di riga ... è equivalente a IF...THEN solo che IF...GOTO deve essere seguito sia dal numero di una riga, sia da un altro statement.

IF...THEN Es. IF X < 0 THEN PRINT "X MINORE DI 0"
IF espressione THEN numero di riga

Salta allo statement specificato se la relazione è vera.

Esegue tutti gli statements sulla riga dopo il THEN se la relazione è vera.

Es. 25 IF X = 5 THEN 50: Z = A

ATTENZIONE: Lo "Z = A" non verrà mai eseguito, perchè se la relazione è vera il BASIC salta alla riga 50, e se è falsa prosegue l'elaborazione alla riga successiva alla riga 25.

Es. 26 IF X < 0 THEN PRINT "ERRORE, X NEGATIVO": GOTO 350

In questo esempio, se X è minore di 0, viene eseguito lo statement PRINT e poi lo statement GOTO farà proseguire l'esecuzione dalla riga 350. Se X è maggiore o uguale a 0, il BASIC eseguirà la riga dopo la 26.

LET Es. 300 LET W = X
Assegna un valore a una variabile.
LET è opzionale.

Es. 310 V = 5.1

NEXT Es. 340 NEXT V
NEXT (variabile)
Segna la fine di un anello FOR.

Es. 345 NEXT

Se non viene specificata la variabile si intende quella dell'anello FOR piú recente.

Es. 350 NEXTV,W

Un singolo NEXT può venir usato per chiudere più statement FOR.
Equivale a NEXT V: NEXT W.

ON...GOSUB Es. 110 ON I GOSUB 50, 60
On espressione GOSUB riga (,riga)...
Identico a "ON...GOTO", solo che viene eseguita la chiamata a una subroutine (GOSUB) invece di un GOTO.
RETURN, nella subroutine, fa ritronare allo statement seguente l'ON...GOSUB.

ON...GOTO Es. 100 ON I GOTO 10, 20, 30, 40
ON espressione GOTO riga (,riga)...
Salta alla riga indicata dall'I-esimo numero dopo il GOTO, cioè:
IF I = 1 GOTO 10
IF I = 2 GOTO 20
IF I = 3 GOTO 30
IF I = 4 GOTO 40
Se I = 0 o I cerca di indicare una riga inesistente (= 5 in questo caso) viene eseguito lo statement seguente lo statement ON. Se però I è 255 o 0, viene segnalato un errore FC.
Un ON...GOTO può essere seguito da tanti numeri quanti ce ne stanno su una riga.

Es. 105 ON SGN (X) + 2 GOTO 40, 50, 60

Questo statement fa saltare alla riga 40 se l'espressione X è minore di zero, alla riga 50 se è uguale a zero, e alla riga 60 se è maggiore di zero.

REM Es. 500 REM ADESSO V = 0
REM qualsiasi scritta.
Permette al programmatore di mettere commenti nei suoi programmi. Gli statement REM non vengono eseguiti, ma vi si può saltare. Uno statement REM termina con la fine della riga, e non con un ":". In questo caso il V = 0 non verrà mai eseguito dal BASIC.
Es. 500 REM METTO V = 0:V = 0
Es. 505 V = 0: REM METTO V = 0
In questo caso il V = 0 viene eseguito.

RESTORE Es. 510 RESTORE
Permette di rileggere gli statements DATA. Dopo un RESTORE, il primo dato del primo statement DATA del programma. Il secondo dato letto sarà secondo del primo statement, e cosí via, come in una normale operazione di READ.

RETURN Es. 50 RETURN
Fa ritornare da una subroutine allo statement successivo all'ultimo GOSUB eseguito.

STOP Es. 900 STOP
Arresta l'esecuzione di un programma e fa riprendere il modo di funzionamento a comandi diretti.
Stampa BREAK IN LINE 900 (in questo esempio. Significa "Arresto alla riga 900). CONT dopo uno STOP fa riprendere l'esecuzione dallo statement seguente lo STOP.

USR Es. 200 V = USR (I)
USER (argomento).
Viene chiamata con argomento I, la subroutine scritta in linguaggio macchina. Vedi POKE, PEEK e l'appendice F.

WAIT Es. 805 WAIT I,J,K
806 WAIT I,J
WAIT indirizzo, maschera (selezione). Questo statement legge il contenuto della locazione I, ne fa un OR esclusivo con la maschera di selezione K, e poi un AND con la maschera J. Questa sequenza viene ripetuta finchè non viene ottenuto un risultato diverso da zero. Dopodichè l'elaborazione riprende dallo statement seguente il WAIT. Se lo statement WAIT ha solo due argomenti, K viene assunto uguale a zero. Se aspettate che un bit vada a zero, dovete mettere un I nella posizione corrispondente di K.
Se la selezione K e la maschera J devono essere = 0 e = 255. L'indirizzo I deve essere compreso tra 0 e 65535.

STATEMENT DI INGRESSO/USCITA DEL BASIC

STAT.	SINTASSI/FUNZIONE
DATA	Es. 10 DATA, 3, -1 E 3,.04 DATA dato (,dato...) Specifica i dati e viene letto da sinistra a destra. Le informazioni appaiono negli statements di dati dello stesso ordine in cui verranno lette nel programma. Negli statement di dati possono comparire anche espressioni. Es. 20 DATA "FOO", Z1 Gli statement DATA possono contenere anche stringhe. Se nella stringa volete includere anche spazi iniziali, due punti (:) o virgole (,), la stringa deve essere messa tra Virgolette. É impossibile avere le virgolette tra i dati o i caratteri di una stringa. ("''BASIC''" non è ammesso).
INPUT	Es. 3 INPUT V,W,W2 INPUT (messaggio da stampare), variabile, (,variabile). Richiede i dati dal terminale (devono venir battuti). Ogni valore deve essere separato dal precedente da una virgola (,). L'ultimo valore battuto deve essere seguito da un ritorno carrello. Come carattere di avvertimento viene stampato un "?". Come risposta a uno statement INPUT possono venire battute solo costanti, come 4.5 E-3 o "GATTO". Se vengono battuti meno dati di quelli richiesti, viene stampato un "??". Bisogna allora battere gli altri dati. Se invece vengono battuti piú dati di quelli richiesti, viene segnalato un "EXTRA DATA". Le stringhe vanno introdotte nel medesimo formato già specificato per i DATA. Es. 5 INPUT "VALORE", V Opzionalmente, viene stampato un messaggio ("VALORE") prima di richiedere i dati dal terminale. Se come risposta a uno statement di ingresso, viene battuto RETURN, il BASIC ritorna al modo di funzionamento a comandi diretti. Se, dopo aver interrotto l'esecuzione di uno statement INPUT, viene battuto CONT, l'esecuzione riprende allo statement INPUT. Es. 15 INPUT! "VALORE"; V Se dopo INPUT viene messo il carattere opzionale (!), i messaggi e i dati battuti dall'utente vengono stampati (anche se la stampante è disabilitata) e visualizzati.

PRINT

Es. 360 PRINT X,Y:Z
370 PRINT "   "
380 PRINT X,Y;
390 PRINT

PRINT (!) espressione (,espressione)

Stampa il valore dell'espressione sul terminale. Se la lista dei valori da stampare non termina con una virgola (,) o con un punto e virgola (;), viene stampato un ritorno carrello/nuova riga al termine della stampa dei valori.

Es. "IL VALORE È"; A

400 PRINT A2, B

Si possono stampare anche stringhe racchiuse tra virgolette ("). Se punto e virgola (;) separa due espressioni nell'elenco, i loro valori vengono stampati uno accanto all'altro. Se compare una virgola dopo un'espressione dell'elenco e la testina di stampa è alla posizione 51 o oltre, viene eseguito un ritorno carrello: nuova riga. Se il carrello invece si trova prima della posizione 51, vengono stampati degli spazi finchè il carrello si trova all'inizio del successivo cambio di 10 colonne. Se c'è una stringa di spazi racchiusi tra virgolette come nella riga 370 degli esempi, viene stampato un ritorno carrello / nuova riga.

POS

Es. 260 PRINT POS (I)

POS (espressione).

Dà la posizione del cursore sul display. La posizione piú a sinistra del display è la posizione zero.

Bisogna usare un operando formale: 0 oppure 1.

READ

Es. 490 READ V,W

READ variabile (,variabile)....

Legge i dati di uno statement DATA e li assegna alle variabili specificate. Il primo dato letto è il primo dato elencato nel primo statement DATA del programma.

Il secondo è il secondo dato del primo statement DATA, e cosí via. Quando sono stati letti tutti i dati del primo statement DATA si passa al primo elemento del secondo statement DATA del programma. Un tentativo di leggere piú dati di quelli che ci sono negli statement DATA viene segnalato come errore di tipo OD (out of data = fuori dai dati).

SPC

Es. 250 PRINT SPC (I)

Stampa I spazi sul terminale. Può essere usato solo in uno statement PRINT. I deve essere > = 0 e < = 255, altrimenti viene segnalato un errore del tipo FC.

TAB — Es. 240 PRINT TAB(I)
TAB (espressione).

Fa avanzare il carrello della stampante fino alla posizione (colonna) I.

Può essere usato solo in uno statement PRINT. Zero è la colonna piú a destra 59 la piú a sinistra. Se il carrello è oltre la posizione I, non viene effettuato alcuno spostamento. I deve essere > = 0 e < = 255.

FUNZIONI DEL BASIC CHE OPERANO SU STRINGHE

ASC

Es. 300 PRINT ASC (X$)
ASC (espressione).

Calcola il valore numerico ASCII del primo carattere dell'espressione di tipo stringa X$. V. l'appendice E per la tavola delle conversioni ASCII/numeri. Se X$ è una stringa vuota, viene segnalato un errore FC.

CHR$

Es. 275 PRINT CHR$(I)
CHR$ (espressione).

Costruisce una stringa il cui solo carattere è l'equivalente ASCII del valore dell'argomento I, che deve essere > = 0 e < = 255. V. l'appendice E.

LEFT$

Es. 310 PRINT LEFT$ (X$,I)
LEFT$ (stringa, lunghezza).

Dà gli I caratteri piú a sinistra dell'espressione stringa X$. Se I = 0 o 255 viene segnalato un errore FC.

LEN

Es. 220 PRINT LEN
LEFT$ (stringa, lunghezza).

Dà la lunghezza in caratteri dell'espressione di tipo stringa X$. Anche gli spazi e i caratteri non di stampa vengono considerati parte della lunghezza.

MID$

Es. 330 PRINT MIDS (X$,I)
MID$ (stringa, inizio, lunghezza).

MID$ chiamata con due argomenti dà i caratteri dell'espressione di tipo stringa a X$ a partire dalla posizione I. Se I LEN (X$) MID$ ritorna una stringa vuota (di lunghezza zero). Se I = 0 o 255, viene segnalato un errore FC.

Es. 340 PRINT MID$ (X$,I,J)

MID$ chiamata con tre argomenti forma una stringa composta dai caratteri dell'espressione X$ a partire dal carattere I-esimo e lunga J caratteri. Se I LEN (X$), MID$ ritorna una stringa vuota. Se I o J = 0 o 255, viene segnalato un errore FC.

Se J specifica piú caratteri di quanti ce ne siano dalla I-esimo all'ultimo, vengono presi tutti i caratteri dall'I-esimo in poi.

RIGHT$ Es. 320 PRINT RIGHT$ (X$,I)
RIGHT$ (stringa, lunghezza).

Dà gli I caratteri piú a destra dell'espressione X$. Se I = 0 o 255, viene segnalato un errore FC.

Se I LEN (XS), RIGHTS dà tutta la stringa X$.

STR$ Es. 290 PRINT STR$ (X)

Dà una stringa che è la rappresentazione in caratteri dell'espressione numerica X. per esempio, STR$ (3.1) = "3.1".

VAL Es. 280 PRINT VAL (X$)

Ritorna l'espressione di tipo stringa X$ convertita in numero. Per esempio, VAL ("3.1") = 3.1. Se il primo carattere della stringa che non sia uno spazio non è un piú (+), un meno (-) una cifra o un punto decimale (.); ritorna il valore zero.

FUNZIONI ARITMETICHE DEL BASIC

STAT.	SINTASSI/FUNZIONE
ABS	Es. 120 PRINT ABS (X) ABS (espressione) Dà il valore assoluto dell'espressione X. Se X =0, ABS dà X, altrimenti dà -X.
ATN	Es. 210 PRINT ATN(X) ATN (espressione). Dà l'arcotangente dell'espressione X. Il risultato viene dato in radianti, ed è compreso tra 0 e = 1.5708). Se intendete usare questa funzione, dovete memorizzare il programma per calcolarla. V. l'appendice H.
COS	Es. 200 PRINT COS(X) COS (espressione) Dà il coseno dell'espressione X;X viene interpretato come se fosse in radianti.
EXP	Es. PRINT EXP(X) EXP (espressione) Dà la costante "e" (2.71828) elevata alla potenza X (e X). Il massimo argomento che può essere usato come argomento di EXP senza superamento della capacità di rappresentazione (overflow) è 88.0296.
INT	Es. 140 PRINT INT(X) INT (espressione) Dà il massimo intero che sia minore o uguale all'espessione X. Per esempio: INT (.23) = 0, INT (7) = 7, INT (-1) = -1, INT (-2) = -2, INT (1.1) = 1. In questo modo si può arrotondare X a D posizioni decimali: INT (X * 10 D + .5)/10 ID
LOG	Es. 160 PRINT LOG (X) LOG (espressione) Dà il logaritmo naturale (in base e) dell'espressione X. Per ottenere il logaritmo in base Y di X usate la formula LOG (X)/LOG(Y). Es. Il logaritmo in base 10 di 7 è = LOG (7)/LOG(10).

RND Es. 170 PRINT RND(X)
RND (parametro).
Genera un numero casuale compreso tra 0 e 1. Il parametro X controlla la generazione del numero in questo modo: X <0 comincia una nuova sequenza di numeri casuali. X=0 dà l'ultimo numero casuale tra 0 e 1.
Notate che (B - A) * RND (I) + A genera un numero casuale tra A e B.

SGN Es. 230 PRINT SGN
SGN (espressione).
Dà 1 se X>0, 0 se X=0 e -1 se X<0.

SIN SIN (espressione)
Dà il senso dell'espressione X. X va dato in radianti. Notate che COS(X)= SIN (X+3.14159/2) e che I radiante = 180/gradi=57.2958 gradi. Perciò il seno di X gradi=SIN (X/57.2958).

SQR Es. 180 PRINT SQR(X)
SQR (espressione)
Dà la radice quadrata dell'espressione X. Se X è negativo, viene segnalato un errore FC.

TAN Es. 200 PRINT TAN(X)
TAN (espressione)
Dà la tangente dell'espressione X. X in radianti.

PGR Costante = 3.14159266 esprime il valore di pigreco.
Es. 100 PRINT PGR.

ULTERIORI FUNZIONI OTTENIBILI.

Le seguenti funzioni, sebbene non comprese nel BASIC, possono venire calcolate usando le funzioni già esistenti nel BASIC.

FUNZIONE	FUNZIONE ESPRESSA IN TERMINI DI FUNZIONI DEL BASIC
SECANTE	SEC(X) = 1/COS(X)
COSECANTE	CSC(X) = 1/SIN(X)
COTANGENTE	COT(X) = 1/TAN(X)
ARCOSENO	ARCSIN(X) = ATN(X/SQR(-X X + 1))
ARCOSENO*	ARCCOS(X) = -ATN(X/SQR(-X X + 1)) + 1.5708
ARCOSECANTE*	ARCOSEC(X) = ATN(SQR(X X-1)) + (SGN(X)-1 1.5708
ARCOCOSECANTE*	ARCCSC(X) = ATN(1/SQR(X X-1)) + (SGN(X)-1)1.5708
ARCOCOTANGENTE*	ARCCOT(X) = -ATN(X) + 1.5708
SENO IPER. *	SINH(X) = (EXP(X)-EXP(-X))/2
COSENO IPER.	COSH(X) = (EXP(X) + EXP(-X))/2
TANGENTE IPER.	TANH(X) = -EXP(-X) / (EXP(X) + EXP(-X))2 + 1
SECANTE IPER.	SECH(X) = 2/(EXP(X) + EXP(-X)
COSECANTE IPER.	CSCH(X) = 2/(EXP(X)-EXP)(-X)
COTANGENTE IPER.	COTH(X) = EXP(-X)/(EXP(X)-EXP(-X))2 + 1
ARCOSENO IPER.	ARGSINH(X) = LOG(X + SQR(X X + 1))
ARCOCOSENO IPER.	ARGCOSH(X) = LOG(X + SQR(X X-1))
ARCOTANGENTE IPER.	ARGTANH(X) = LOG(1 + X) / (1-X))/2
ARCOSECANTE IPER.	ARGSECH(X) = LOG((XQR(-X X + 1) + 1)/X)
ARCOCOSECANTE IPER.	ARGCSCH(X) = LOG(SGN(X)SQRT(X X + 1) + 1)/X)
ARCOCOTANGENTE IPER.	ARGCOTH(X) = LOG((X + 1) / (X-1))/2

*Queste funzioni richiedono la funzione ATN, definita dall'utente. Vedere l'appendice G per i dettagli.

APPENDICE A
MESSAGGI DI ERRORE

Al verificarsi di un errore, il BASIC ritorna al livello dei comandi diretti e stampa OK. I valori delle variabili e il testo del programma rimangono intatti, ma l'esecuzione del programma non può venire continuata e lo stato dei GOSUB e dei FOR viene perso.

Se si verifica un errore durante l'esecuzione di uno statement diretto; non viene stampato alcun numero di riga.

Formato dei messaggi di errore:
Statement diretti:? XX ERROR
Statement indiretti: ? XX ERROR IN YYYY

In entrambi gli esempi "XX" è il codice dell'errore. "YYYY" è il numero della riga dello statement indiretto in cui si è verificato l'errore.

Questi sono codici di errore e i loro significati:

CODICE DI ERRORE	SIGNIFICATO
BS	Bas Subscript (= indice sbagliato). È stato fatto un tentativo di operare con un elemento di una matrice che si trova fuori dalle dimensioni della matrice stessa. Questo errore può verificarsi se ci si riferisce a una matrice con un numero sbagliato di dimensioni; per esempio, LET A (I,I,I) = Z quando A è stata dimensionata DIM A (2,2).
CN	Errore di continuazione. Si è tentato di continuare un programma inesistente, o dopo il verificarsi di un errore, o dopo una modifica del programma.
DD	Doppio dimensionamento. Dopo che una matrice è stata dimensionata, viene incontrato un altro statement di dimensionamento per la stessa matrice. Questo errore si verifica solitamente se a una matrice viene assegnata automaticamente una dimensione di 10 a causa di uno statement tipo A(I) = 3, e piú avanti nel programma si incontra uno statement tipc DIM A (100).
FC	Function Call (= chiamata di funzione). Il parametro passato a una funzione matematica o operante su stringhe era fuori dal campo. Gli errori FC si verificano a causa di: 1. un indice di una matrice negativo (LET A(-1) = 0) 2. l'indice di una matrice troppo grande (32767)

3. l'argomento di un LOG negativo o zero.
4. l'argomento di un SQR negativo
5. A B con A negativo e B non intero.
6. una chiamata a USR prima di aver specificato l'indirizzo della subroutine in linguaggio macchina.
7. una chiamata a MID$, LEFT$, RIGHT$, WAIT, PEEK, POKE, TAB, SPC, o ON... GOTO con un argomento improprio.

ID — Illegal Direct (= comando diretto illegale). Non è permesso usare gli statement INPUT, DEF o GET come comandi diretti.

LS — Long String (= stringa lunga). Usando l'operatore di concatenamento si è tentato di costruire una stringa di piú di 255 caratteri.

NF — NEXT senza FOR. La variabile in uno statement NEXT non corrisponde ad alcun statement FOR precedentemente eseguito.

OD — Out of DATA (= fuori dai dati). È stato eseguito un READ ma tutti gli statement DATA sono già stati letti. Il programma ha cercato di leggere troppi dati, oppure non sono stati messi abbastanza dati nel programma.

OM — Out of Memory (= fuori dalla memoria). Il programma è troppo lungo, ci sono troppe variabili, troppi anelli FOR, tropi GOSUB, una espressione troppo complicata o una qualsiasi combinazione di questi (V. l'appendice B).

OV — Overflow. Il risultato di un calo è troppo grande per essere rappresentato nel formato numerico del BASIC. Se si verifica un "underflow" (= numero troppo piccolo, in modulo) viene dato zero come risultato, e l'esecuzione continua senza che alcun messaggio venga stampato.

SN — Syntax Error (= errore di sintassi). Manca una parentesi in una espressione, c'è un carattere illegale in una riga, la punteggiatura non è corretta, ecc.

RG	RETURN senza GOSUB. È stato incontrato un RETURN senza che fosse stato eseguito precedentemente un GOSUB.
UF	Undefined Function (= funzione non definita). È stato fatto riferimento a una funzione utente che non è mai stata definita.
US	Undefined Statement (= statement non definito). È stato tentato un GOTO, GOSUB o THEN indicante uno statement che non esiste.
/0	Divisione per zero.
ST	String temporaries. Una espressione di tipo stringa è troppo complessa. Dividetela in due o piú parti piú brevi.
TM	Type Mismatch (= incompatibilità tra tipi). La parte sinistra di uno statement di assegnamento è una variabile numerica e quella destra una stringa o viceversa, oppure una funziona che richiede una stringa come argomento viene chiamata un'espressione numerica o viceversa.

APPENDICE B

CONSIGLI PER RISPARMIARE SPAZIO

Per rendere piú breve il vostro programma e risparmiare spazio, possono esservi di utilità questi consigli.

1. mettete piú statements sulla stessa riga. Ogni riga del programma ha una testata di 5 bytes. Due di questi 5 bytes contengono il numero della riga (in binario). Ciò significa che ci vuole sempre lo stesso numero di bytes, qualunque sia il numero di cifre che compongono il numero della riga (che perciò può variare tra 0 e 63999). Mettere tutti gli statements possibili su una stessa riga riduce il numero di bytes usati dal vostro programma.

2. eliminate gli spazi inutili dal vostro programma, per esempio:

   10 PRINT X, Y, Z
   usa 3 bytes in piú di:
   10 PRINTX, Y, Z

   Nota: tutti gli spazi tra il numero della riga e il primo carattere che è uno spazio vengono ignorati.

3. Cancellate tutti gli statement REM. Ogni statement REM usa almeno un byte, oltre ai bytes necessari per memorizzare il testo. Per esempio, lo statement 130 REM THIS IS A COMMENT (questo è un commento) richiede 24 bytes di memoria.
   Nello statement 140X = X + Y:REM UPDATE SUM (= aggiorna la somma) il REM usa 14 bytes di memoria, compreso il due punti prima del REM.

4. Usate variabili invece delle costanti. Supponete di usare la costante 4.14578 dieci volte nel vostro programma. Se inserite uno statement 10 0 = 4.14578 nel programma e usate 0 invece di 4.14578 ogni volta che ce n'è bisogno, risparmierete 40 bytes. Inoltre avrete anche un aumento della velocità.

5. Non è necessario che un programma finisca con un END, perciò si può cancellare uno statement END alla fine del programma.

6. Riusate le stesse variabili. Se avete una variabile T che viene usata per contenere un risultato provvisorio in una parte del programma, e poi avete bisogno di una variabile provvisoria piú avanti nel programma riusatela. Oppure se chiedete all'utente del terminale di rispondere SI o NO a due domande diverse in due momenti diversi durante l'esecuzione del programma, usate la stessa variabile provvisoria A$ per memorizzare la risposta.

7. Usate GOSUB per eseguire porzioni del programma che compiono le stesse azioni.

8. Usate l'elemento zero delle matrici; per esempio A(O), B(O,X).

INFORMAZIONE SULL'ALLOCAZIONE DI MEMORIA

Le variabili numeriche semplici (non le matrici) come V usano 7 bytes: 2 per il nome della variabile e 5 per il valore. Anche le variabili del tipo stringa semplici richiedono 7 bytes: 2 per il nome, 1 per la lunghezza, 2 per un puntatore, e 2 inutilizzati.

Le matrici di variabili richiedono 7 bytes di testata, oltre allo spazio necessario a contenere ogni elemento della matrice. Ogni elemento che sia un numero intero richiede 2 bytes; se invece gli elementi sono variabili di tipo stringa o variabili in virgola mobile si richiedono 3 e 5 bytes rispettivamente.

Le variabili di tipo stringa usano anche un byte per ogni carattere componente la stringa. Ciò vale sia per le variabili stringa semplici, come A$, sia per gli elementi di matrici di stringhe, come 01$ (5,2). Per memorizzare la definizione di una nuova funzione definita con uno statement DEF si richiedono 7 bytes.

Le parole riservate come FOR, GOTO, o NOT, e i nomi delle funzioni intrinseche come COS, INT e STR$ richiedono un solo byte di memoria di programma. E tutti gli altri caratteri nei programmi richiedono un byte di memoria ciascuno.

Durante l'esecuzione di un programma, lo spazio viene così allocato dinamicamente sullo stack:

1. Ogni anello FOR...NEXT attivo usa 22 bytes.

2. Ogni GOSUB attivo (cioè, il cui RETURN non è stato ancora eseguito) usa 6 bytes.

3. Ogni parentesi incontrata in una espressione usa 4 bytes, e ogni risultato provvisorio ottenuto durante il calcolo di una espressione usa 12 bytes.

CONSIGLI PER AUMENTARE LA VELOCITÀ

Seguendo questi consigli riuscirete ad aumentare la velocità di esecuzione del vostro programma in BASIC. Notate che alcuni di questi consigli sono gli stessi già dati per ridurre l'occupazione di memoria.

Ciò significa che in molti casi potrete aumentare l'efficienza del vostro programma sia per quanto riguarda la velocità che le dimensioni.

1. Cancellate tutti gli spazi non necessari e i REM del programma. Questo porta a una piccola riduzione del tempo di esecuzione perchè altrimenti il BASIC dovrà ignorare o saltare gli spazi e i REM.

2. QUESTO PROBABILMENTE È IL CONSIGLIO PIÙ IMPORTANTE PER AUMENTARE LA VELOCITÀ.
   Usate le variabili al posto delle costanti. Ci vuole più tempo a convertire una costante nella sua rappresentazione in virgola mobile che a prendere il valore di una variabile semplice o di un elemento della matrice. Questo è molto importante soprattutto all'interno di anelli FOR...NEXT o altri pezzi di programma ripetuti spesso.

3. Le variabili che vengono incontrate per prime durante l'esecuzione di un programma in BASIC vengono messe all'inizio della tabella delle variabili. Ciò significa che uno statement del tipo:
   5 A=0: B=A: C=A, assegnerà ad A il primo, a B il secondo, e a C il terzo posto nella tabella dei simboli (semprechè la riga 5 sia il primo statement del programma ad essere eseguito). Più avanti nel programma, quando il BASIC incontra un riferimento alla variabile A, dovrà esaminare un solo elemento della tabella dei simboli per trovare A, due per trovare B, tre per trovare C, ecc.

4. Usate gli statement NEXT senza la variabile indice. NEXT è un po' più veloce di NEXT I, perchè non viene fatto alcun controllo per vedere se la variabile specificata nel NEXT è la stessa variabile dello statement FOR eseguito più recentemente.

APPENDICE D

TRADUZIONE DI PROGRAMMI BASIC NON SCRITTI PER IL BASIC DEL 6500

Benchè le implementazioni del BASIC su computer diversi siano per molti versi simili, ci sono alcune incompatibilità alle quali bisogna stare attenti nel convertire i programmi in BASIC non scritti in 6500 BASIC.

1. Indici delle matrici. Alcuni BASIC usano ""e"" per racchiudere gli indici delle matrici. Il nostro BASIC usa "("e")".

2. Stringhe. Parecchi BASIC esigono il dimensionamento (dichiarato) della lunghezza delle stringhe prima del loro uso. Tutti gli statement di dimensionamento di questo tipo vanno eliminati dal programma.
   In alcuni di questi BASIC poi una dichiarazione del tipo DIM A$ (I,J) dichiara una matrice di stringhe composta da J elementi ciascuno di lunghezza I. Gli statement DIM di questo tipo vanno convertiti così:

   DIM A$ (J).

   Questo BASIC per il concatenamento delle stringhe usa " + ", non "," nè "&".
   Questo BASIC usa LEFT$, RIGHT$ e MID$ per prelevare pezzi di stringhe. Altri BASIC usano A$(I) per prendere l'I-esimo carattere della stringa A$, e A$ (I,J) per prendere il pezzo della stringa A$ compreso tra l'I-esimo e il J-esimo carattere. Convertite così:

A$(I)	equivale a	MID$ (A$, I, 1)
A$ (I,J)	equivale a	MID$ (A$, I,J-I + 1)

3. Assegnamenti multipli. Alcuni BASIC permettono statement del tipo: 500 LET B = C = O. Questo statement ha l'effetto di dare alle variabili B e C il valore zero.
   Nel nostro caso questo ha un effetto totalmente diverso. Tutti gli " = " alla destra del primo vengono interpretati come operatori logici di confronto.
   Quindi se C fosse uguale a 0, a B verrebbe assegnato il valore -1, altrimenti il valore 0. Il modo piú semplice per convertire gli statement di questo tipo è quello di riscriverli cosí:

   500 C = 0 : B = 0

4. Alcuni BASIC usano "/" invece di ":" per separare gli statements su una stessa riga. Cambiate le "/" in ":" nel programma.

5. I programmi che usano funzioni matematiche disponibili in alcuni BASIC dovranno essere riscritti usando anelli FOR-NEXT per compiere le operazioni appropriate.

6. Uno statement PRINT senza argomenti non fa avanzare la carta della stampante. Per far avanzare la carta (per ottenere, cioè, una riga bianca) usate PRINT " " (spazio).

APPENDICE E

CODICE DEI CARATTERI ASCII

Decimale	Carat.	Decimale	Carat.	Decimale	Carat.
000	NUL	043	+	086	V
001	SOH	044	,	087	W
002	STX	045	-	088	X
003	ETX	046	.	089	Y
004	EOT	047	/	090	Z
005	ENQ	048	0	091	[
006	ACK	049	1	092	\
007	BEL	050	2	093	]
008	BS	051	3	094	↑
009	HT	052	4	095	←
010	LF	053	5	096	'
011	VT	054	6	097	a
012	FF	055	7	098	b
013	CR	056	8	099	c
014	SO	057	9	100	d
015	SI	058	:	101	e
016	DLE	059	;	102	f
017	DC1	060	<	103	g
018	DC2	061	=	104	h
019	DC3	062	>	105	i
020	DC4	063	?	106	j
021	NAK	064	@	107	k
022	SYN	065	A	108	l
023	ETB	066	B	109	m
024	CAN	067	C	110	n
025	EM	068	D	111	o
026	SUB	069	E	112	p
027	ESCAPE	070	F	113	q
028	FS	071	G	114	r
029	GS	072	H	115	s
030	RS	073	I	116	t
031	US	074	J	117	u
032	SPACE	075	K	118	v
033	!	076	L	119	w
034	"	077	M	120	x
035	#	078	N	121	y
036	$	079	O	122	z
037	%	080	P	123	{
038	&	081	Q	124	!
039	'	082	R	125	}
040	(	083	S	126	~
041	)	084	T	127	DEL
042	*	085	U		

LF = Line Feed = nuova riga
FF = Form Feed
CR = Ritorno carrello
DEL = Rubout sulla telescrivente

APPENDICE F

SUBROUTINES IN LINGUAGGIO MACCHINA

Il BASIC dell'AMICO 2000 permette all'utente l'uso di subroutines in linguaggio macchina, mediante l'uso della funzione USR (W). Questa funzione permette il passaggio di un solo parametro tra il BASIC e la subroutine.

Per prima cosa bisogna allocare la memoria sufficiente per la subroutine. Il BASIC, se l'utente non specifica diversamente, usa tutta la memoria RAM a partire dalla locazione decimale 16402 (4012 in esadecimale). Potete limitare l'uso di memoria del BASIC rispondendo alla domanda MEMORY SIZE? (V. paragrafo 100) con un numero minore di 20480.

In questo modo potete lasciare abbastanza spazio per la subroutine nella parte alta della RAM. Per esempio, se la vostra risposta a MEMORY SIZE? è 18432, vi resteranno a disposizione gli ultimi 2048 bytes di RAM per la subroutines in linguaggio macchina, questo se disponete di soli 4 kKbyte di RAM.

Il parametro (W), passato alla subroutine da USR (W), viene convertito in virgola mobile e messo nell'accumulatore situato in $A9. La struttura dell'accumulatore è la seguente.

$A9	Esponente + $81 ($80 se la mantissa = 00)
$AA-$AD	Mantissa normalizzata in modo che il bit 7 del byte piú significativo è $AA; $AD è il meno significativo (LSB)
$AE	Segno della mantissa.

Un parametro passato a una subroutine in linguaggio macchina dal BASIC, può essere troncato dalla subroutine in un intero di 2 byte e messo in $AC (MSB) e $AD (LSB).

Se il parametro è maggiore di 32767 o minore di -32768, viene segnalato un errore FC. L'indirizzo della subroutine che converte un numero in virgola mobile in un intero si trova in $2006, $2007.

Il parametro passato della subroutine in linguaggio assemblatore al BASIC va riconvertito in virgola mobile. L'indirizzo della subroutine che opera questa conversione si trova in $2008, $2009. Il MSB ($AC) dell'intero deve essere nel registro Y.

Prima di eseguire l'USR, l'indirizzo di partenza della subroutine in linguaggio assemblatore deve venir messo nelle locazioni $04 (LSB) e $05 (MSB). Generalmente si ottiene ciò usando il comando POKE. Notate che è possibile chiamare piú di una subroutine in assembler semplicemente chiamando l'indirizzo di partenza contenuto in $04 e $05.

```
10 POKE 4,0: POKE 5,74
20 INPUT "NUMBER";N
30 X = USR (N)
40 PRINT X
50 GOTO 20
```

Questo è il listing di un programma in BASIC che chiama una subroutine in assembler che comincia alla locazione $4A00. Il funzionamento è il seguente:

Riga 10 - Mette l'indirizzo di partenza della subroutine in assembler ($4A00) nelle locazioni $04 e $05, usando POKE.

Riga 20 - Chiede un numero "N"

Riga 30 - Chiama la subroutine, con N come argomento.

Riga 40 - Al ritorno della subroutine, il programma stampa X, l'argomento passato dalla subroutine al programma BASIC.

Riga 50 - Ritorna indietro per chiedere un nuovo "N".

APPENDICE G

IMPLEMENTAZIONE DELL'ATN

La funzione ATN (arcotangente) può venire programmata nella RAM usando il comando M (Memory) dell'AMICO 2000 e può essere allocata in una qualsiasi zona che non sia la pagina zero e la pagina 1 (0000 ÷ 01FF).

Una zona libera adatta all'uso è la pagina 2 quindi a partire dalla locazione 0200 inseriremo il seguene programma.

0200	0B	76	B3	83	BD	D3	79	1E
0208	F4	A6	F5	7B	83	FC	B0	10
0210	7C	0C	1F	67	CA	7C	DE	53
0218	CB	C1	7D	14	64	70	4C	7D
0220	B7	EA	51	7A	7D	63	30	88
0228	7E	7E	92	44	99	3A	7E	4C
0230	CC	91	C7	7F	AA	AA	AA	13
0238	81	00	00	00	00	00	00	00
0240	A5	AE	48	10	03	20	B7	3C
0248	A5	A9	48	C9	81	90	07	A9
0250	FA	A0	36	20	4D	38	A9	00
0258	A0	02	20	43	3D	68	C9	81
0260	90	07	A9	59	A0	3E	20	8E
0268	35	68	10	03	4C	B7	3C	60

Per comodità riportiamo di seguito la codifica del programma in modo piú leggibile.

TABELLA VALORI

```
0200   0B
  .
  .
  .
023F   00
0240   A5   AE     ATN    LDA $AE    Carico il segno
0242   48                 PHA        Lo salvo
0243   10   03            BPL ATN1   Testo segno
0245   20   B73C          JSR $3CB7
0248   A5   A9     ATN1   LDA $A9    Carico l'esponente
024A   48                 PHA        Lo salvo
024B   C9   81            CMP #$81   Comparo
024D   90   07            BCC ATN2   e testo
024F   A9   FA            LDA #$FA   Presetto valori
0251   A0   36            LDY #$36
0253   20   4D38          JSR $384D  Eseguo
0256   A9   00     ATN2   LDA #$00   Presetto valori
0258   A0   02            LDY #$02   della tabella
025A   20   433D          JSR $3D43  Eseguo
025D   68                 PLA        Recupero valore
025E   C9   81            CMP #$81   Comparo
0260   90   07            BCC ATN3   Testo
0262   A9   59            LDA #$59   Presetto valori
0264   A0   3E            LDY #$3E
0266   20   8E35          JRS $358E  Eseguo
0269   68                 PLA        Recupero
026A   10   03            BPL ATN4   Testo
026C   4C   B73C          JMP $3CB7  Eseguo
026F   60          ATN4   RTS        Ritorno
```

Questo programma può essere salvato su cassetta e ricaricato tutte le volte che desideriamo utilizzare la funzione ATN.

Prima di potere usare la funzione è però necessario modificare l'indirizzo di JUMP che ritrova alle locazioni BC e BD di pagina base cosa che può essere convenientemente fatta usando il comando POKE come di seguito riportato.

```
10 POKE 188,64
20 POKE 189,02
```

Per verificare poi che tutto giri nel modo corretto possiamo provare ad eseguire la seguente riga di programma:

PRINT ATN (TAN(.5))   il risultato dovrà essere .5